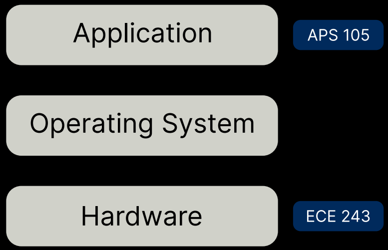
Course Recap

2024 Fall ECE 344: Operating Systems
Lecture 34

Jon Eyolfson

2.0.0

An Operating System Manages Resources



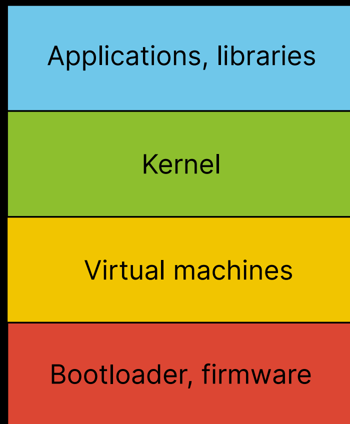
There’s 3 Core Operating System Concepts

Virtualization: share one resource by mimicking multiple independent copies

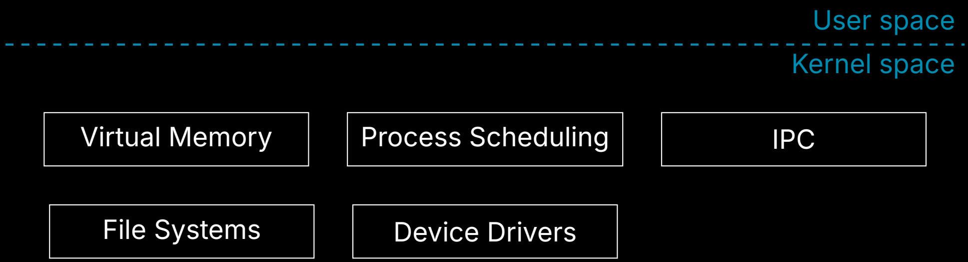
Concurrency: handle multiple things happening at the same time

Persistence: retain data consistency even without power

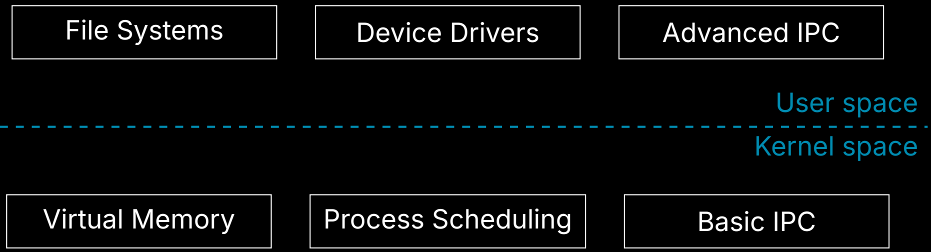
More Privileged CPU Modes Can Access More Instructions



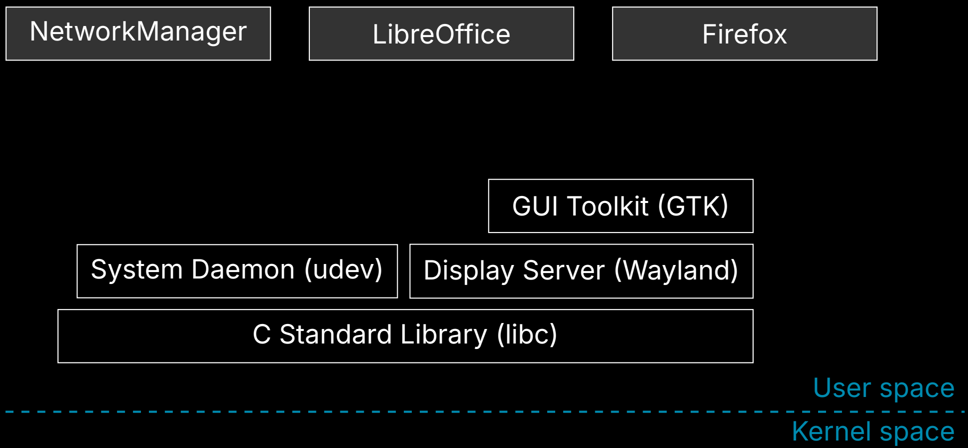
A Monolithic Kernel Runs Operating System Services in Kernel Mode



A Microkernel Runs the Minimum Amount of Services in Kernel Mode



Applications May Pass Through Multiple Layers of Libraries



The Operating System Creates Processes

The operating system has to:

- Maintain process control blocks, including state
- Create new processes
- Load a program, and re-initialize a process with context

You're Responsible for Managing Processes

The operating system maintains a strict parent/child relationship

You should be able to identify (and prevent) the following:

- Zombie processes
- Orphan processes

We Explored Basic IPC in an Operating System

Some basic IPC includes:

- `read` and `write` through file descriptors (could be a regular file)
- Redirecting file descriptors for communcation
- Signals

Signals are like interrupts for user processes
  The kernel has to handle all 3 kinds of “interrupts”

Scheduling Involves Trade-Offs

We looked at few different algorithms:

- First Come First Served (FCFS) is the most basic scheduling algorithm
- Shortest Job First (SJF) is a tweak that reduces waiting time
- Shortest Remaining Time First (SRTF) uses SJF ideas with preemptions
- SRTF optimizes lowest waiting time (or turnaround time)
- Round-robin (RR) optimizes fairness and response time

Scheduling Gets Even More Complex

There are more solutions, and more issues:

- Introducing priority also introduces priority inversion
- Some processes need good interactivity, others not so much
- Multiprocessors may require per-CPU queues
- Real-time requires predictability
- Completely Fair Scheduler (CFS) tries to model the ideal fairness

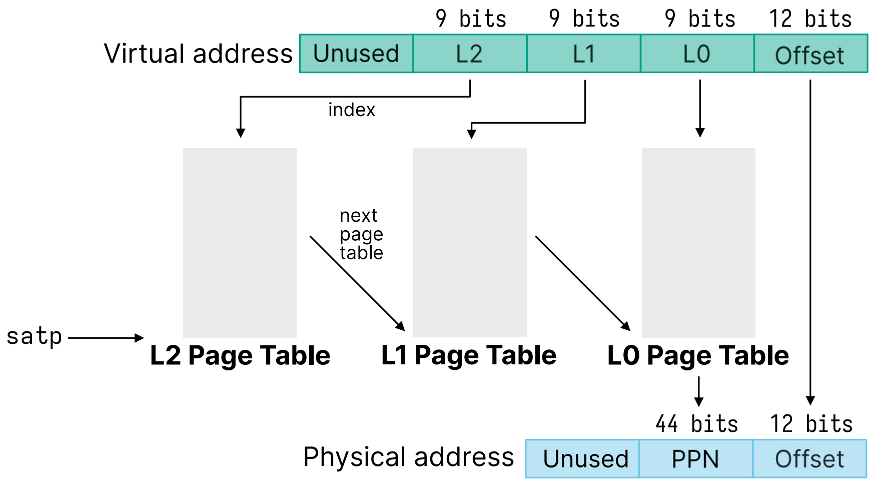
We Use Pages for Memory Translation

Divide memory into blocks, so we only have to translate once per block

Use page tables (array of PTEs) to access the PPN (and flags)

New problem: these page tables are always huge!

Multi-Level Page Tables Save Space for Sparse Allocations



Page Tables Translate Virtual to Physical Addresses

The MMU is the hardware that uses page tables, which may:

- Be a single large table (wasteful, even for 32-bit machines)
- Use the kernel allocated pages from a free list
- Be a multi-level to save space for sparse allocations
- Use a TLB to speed up memory accesses

We Had a Brief Detour

We explored a question using priority scheduling

Also, we looked at `mmap` (not covered on the exam)

Threads Enable Concurrency

We explored threads, and related them to something we already know (processes)

- Threads are lighter weight, and share memory by default
- Each process can have multiple threads (but just one at the start)

Both Processes and (Kernel) Threads Enable Parallelization

- Each process can have multiple (kernel) threads
- Most implementations use one-to-one user-to-kernel thread mapping
- The operating system has to manage what happens during a fork, or signals
- We now have synchronization issues

Another Detour on Sockets, and `ucontext`

You don’t have to know sockets for the exam, but they’re just IPC

`ucontext` is fair game (it’s the state of the user registers)

We Want Critical Sections to Protect Against Data Races

We should know what data races are, and how to prevent them:

- Mutex or spinlocks are the most straightforward locks
- We need hardware support to implement locks
- We need some kernel support for wake up notifications
- If we know we have a lot of readers, we should use a read-write lock

We Used Semaphores to Ensure Proper Order

Previously we ensured mutual exclusion, now we can ensure order

- Semaphores contain an initial value you choose
- You can increment the value using post
- You can decrement the value using wait (it blocks if the current value is 0)
- You still need to be prevent data races

We Explored More Advanced Locking

We have another tool to ensure order

- Condition variables are clearer for complex condition signaling
- Locking granularity matters
- You must prevent deadlocks

Disks Enable Persistence

We explored two topics: SSDs and RAID

- SSDs are more like RAM except accessed in pages and blocks
- SSDs also need to work with the OS for best performance (TRIM)
- Use RAID to tolerate failures and improve performance using multiple disks

Filesystems Enable Persistence

They describe how files are stored on disks:

- API-wise you can open files, and change the position to read/write at
- Each process has a local open file and there’s a global open file table
- There’s multiple allocation strategies: contiguous, linked, FAT, indexed

inodes Are a Hybrid Allocaiton Strategy

inodes are offer greater flexibility over: contiguous, linked, FAT, or indexed

- Everything is a file on UNIX, names in a directory can be hard or soft links

Page Replacement Algorithms Aim to Reduce Page Faults

We saw the following:

- Optimal (good for comparison but not realistic)
- Random (actually works surprisingly well, avoids the worst case)
- FIFO (easy to implement but Bélády's anomaly)
- LRU (gets close to optimal but expensive to implement)

The Clock Algorithm is an Approximation of LRU

Data structures:

- Keeps a circular list of pages in memory
- Uses a reference bit for each page in memory (light grey in next slides)
- Has a “hand” (iterator) pointing to the last element examined

Algorithm, to insert a new page:

- Check the hand’s reference bit, if it’s 0 then place the page and advance hand
- If the reference bit is 1, set it to 0, advance the hand, and repeat

For page accesses, set the reference bit to 1

The Kernel Has To Implement It's Own Memory Allocations

The concepts are the same for user space memory allocation
(the kernel just gives them more contiguous virtual memory pages):

- There's static and dynamic allocations
- For dynamic allocations, fragmentation is a big concern
- Dynamic allocation returns blocks of memory
  - Fragmentation between blocks is external
  - Fragmentation within a blocks is internal
- There's 3 general allocation strategies for different sized allocations
  - Best fit
  - Worst fit
  - First fit

Even More Memory Allocations

The kernel restricts the problem for better memory allocation implementations

- Buddy allocator is a real-world restricted implementation
- Slab allocator takes advantage of fixed sized objects to reduce fragmentation

Virtual Machines Virtualize a Physical Machine

They allow multiple operating systems to share the same hardware

- Virtual machines provide isolation, the hypervisor allocates resources
- Type 2 hypervisors are slower due to trap-and-emulate and binary translation
- Type 1 hypervisors are supported by hardware, IOMMU speeds up devices
- Hypervisors may overcommit resources and need to physically move VM
- Containers aim to have the benefits of VMs, without the overhead

That’s the Course!

Please let me know on Discord what you want me to review in our last lecture